



**ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO**  
**FACULTAD: INFORMÁTICA Y ELECTRÓNICA**  
**ESCUELA DE INGENIERÍA EN SISTEMAS**  
**CARRERA: SOFTWARE**

**GUÍA DE LABORATORIO DE APLICACIONES INFORMÁTICAS II**

**1. DATOS GENERALES:**

**NOMBRE DEL ESTUDIANTE**  
Jeferson Vargas

**CODIGO DEL ESTUDIANTE**  
6473

**FECHA DE REALIZACIÓN:**

2025/07/15

**FECHA DE ENTREGA:**

2025/07/15

**2. OBJETIVO(S):**

**2.1. GENERAL**

- Realizar el análisis de una aplicación (Sistema de facturación) para realizar el proceso de ingeniería inversa.

**2.2. ESPECÍFICOS**

- Analizar los archivos principales de el sistema para entender su funcionamiento
- Realizar el proceso de ingeniería inversa basándonos en los archivos que identificamos como relevantes

**3. METODOLOGÍA**

**Selección del Aplicativo Informático:**

- El primer paso consiste en identificar y seleccionar un aplicativo informático existente sobre el cual se aplicarán las técnicas de ingeniería inversa.
- Se recomienda elegir un software cuya complejidad sea manejable para el alcance de la práctica, pero que a su vez permita la aplicación efectiva de las técnicas.

**Aplicación de la Ingeniería Inversa:**

- Se deben utilizar técnicas de **análisis estático** (sin ejecutar el código) para examinar la estructura del software, como la revisión del código fuente (si está disponible), el uso de desensambladores (como IDA Pro o Ghidra para binarios) y decompiladores (como JD-GUI para Java o dnSpy para .NET) para obtener representaciones de alto nivel del código.
- Adicionalmente, se puede emplear el **análisis dinámico** (observando el comportamiento del software durante su ejecución) utilizando depuradores (como OllyDbg o GDB) para entender los flujos de ejecución, el uso de memoria y las interacciones con el sistema operativo o la red (mediante herramientas como Wireshark).
- El objetivo es comprender a fondo los componentes del aplicativo, sus interrelaciones, los flujos de datos, la lógica de negocio y la arquitectura inferida.

#### 4. EQUIPOS Y MATERIALES:

- Computador
- Entorno integrado de desarrollo (IDE)
- Aula virtual
- Acceso a internet
- Bibliografía

#### 5. ANALISIS DEL SISTEMA

El sistema analizado, del cual se ha examinado un componente clave, tiene como **propósito principal facilitar la emisión y gestión de comprobantes de pago electrónicos, específicamente facturas, de acuerdo con la normativa de la Superintendencia Nacional de Aduanas y de Administración Tributaria (SUNAT) de Perú**. Su función central es automatizar el proceso de creación de documentos fiscales digitales, su firma electrónica y su envío a la autoridad tributaria, garantizando así la validez y el cumplimiento fiscal de las transacciones comerciales.

En esencia, sirve como un **intermediario programático** entre las operaciones de negocio internas y los exigentes requisitos técnicos y legales de la facturación electrónica peruana, transformando datos comerciales en documentos fiscales válidos y comunicándolos eficientemente con SUNAT.

##### Funcionalidades Principales

Basándonos en el código y la estructura observada, las funcionalidades principales de este sistema son:

##### Modelado de Documentos Electrónicos:

- Permite la **estructuración programática de una factura electrónica (Invoice)**, incluyendo todos los campos requeridos por SUNAT como la versión UBL, fechas de emisión y vencimiento, tipo de operación, tipo de documento (01 para factura), serie y número correlativo, moneda (PEN), y campos de observación.
- Capacidad para **definir los detalles de venta (SaleDetail)** de cada ítem de la factura, especificando datos como código de producto, descripción, unidad de medida, cantidad, valores unitarios (con y sin IGV), montos de IGV, y el tipo de afectación al impuesto.
- Soporte para la **inclusión de leyendas (Legend)** en la factura, como la representación textual del monto total.

##### Gestión de Datos Maestros (Empresa y Cliente):

- A través de la clase auxiliar Util y su componente shared, el sistema abstrae la obtención de los **datos de la empresa emisora** (Razon Social, RUC, Dirección, etc.) y los **datos del cliente receptor** (Tipo de Documento, Número de Documento, Nombre/Razón Social, etc.). Esta centralización sugiere una funcionalidad de gestión de entidades previamente cargadas o configuradas.

#### Generación y Envío a la Autoridad Tributaria:

- Orquesta la **conexión con los servicios web de la SUNAT** (o un intermediario como beatose.herokuapp.com) mediante la configuración de un servicio de envío (Greenter\See).
- Realiza el **envío automatizado de la factura** al servicio de SUNAT, lo que implica la generación interna del XML bajo el estándar UBL, la aplicación de la firma digital (gestionada por Greenter), y la transmisión segura del documento.

#### Manejo Básico de Respuestas Fiscales:

- **Verificación del Estado de Envío:** Capacidad para determinar si el documento fue recibido y validado exitosamente por SUNAT.
- **Gestión de Errores:** En caso de fallos en el envío o validación por parte de SUNAT, el sistema captura y muestra el mensaje de error.
- **Procesamiento del CDR:** Si el envío es exitoso, el sistema es capaz de obtener y manejar el Comprobante de Recepción (CDR) emitido por SUNAT, el cual es la constancia oficial de la validez del comprobante.

#### Persistencia de Documentos (Auditoría):

- Utiliza la clase Util para **guardar los archivos XML** de las facturas generadas y los archivos **ZIP que contienen el CDR** recibido de SUNAT. Esta funcionalidad es crucial para el cumplimiento y la auditoría, permitiendo almacenar una copia local de los documentos fiscales emitidos y sus respuestas.

## 6. ARQUITECTURA

El análisis de ingeniería inversa del sistema de facturación, basado en el código fuente proporcionado (especialmente el script de emisión de facturas y las referencias a la clase Util), revela una arquitectura pragmática que combina elementos de un diseño en **Capas** con la aplicación parcial del patrón **Modelo-Vista-Controlador (MVC)**, y la utilización de patrones de diseño específicos como el **Singleton**.

#### Enfoque Arquitectónico Principal: Arquitectura en Capas

El sistema demuestra una clara separación de responsabilidades a través de una arquitectura en capas lógicas:

- **Capa de Orquestación/Controlador:** El script factura-beatose.php se posiciona como una capa de control y orquestación. Su función es recibir la solicitud de emisión de una factura, coordinar la preparación de los datos, interactuar con la lógica de negocio y los servicios de integración, y gestionar la respuesta final. En un sistema más amplio, esta capa podría ser un controlador de una aplicación web o un procesador de un servicio de fondo.
- **Capa de Lógica de Negocio/Dominio:** Esta capa contiene las reglas y el procesamiento central relacionados con la facturación.

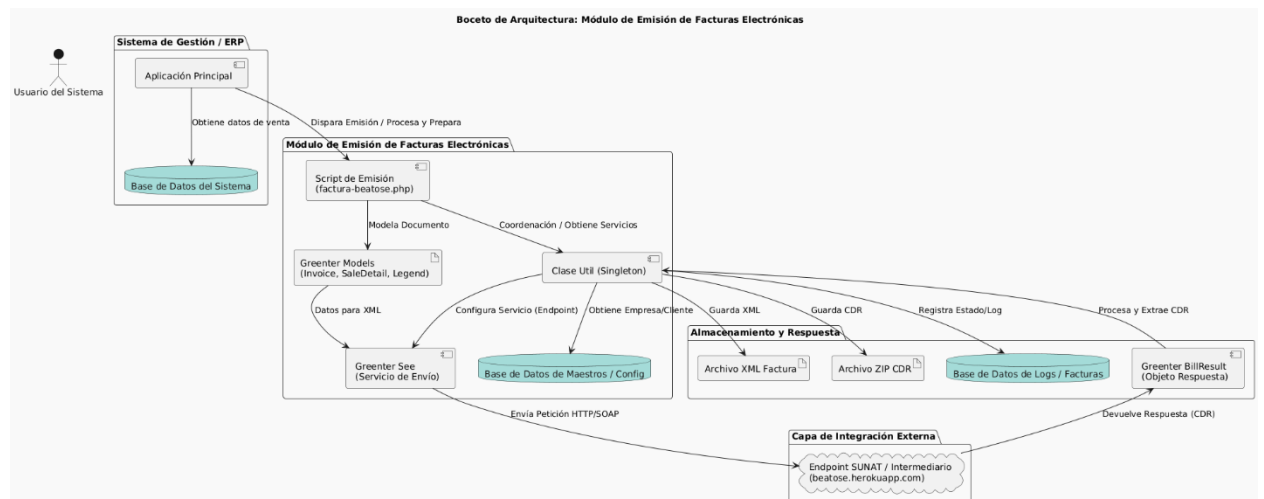
- Los **Modelos de Greenter** (Greenter\Model\Sale\Invoice, Greenter\Model\Sale\SaleDetail, Greenter\Model\Sale\Legend) actúan como representaciones de las entidades de dominio (la factura y sus ítems) y encapsulan la estructura de datos conforme a las especificaciones UBL y SUNAT.
  - La **Clase Util** se identifica como un componente clave en esta capa. Aunque su implementación completa no fue proporcionada, su rol inferido abarca lógica auxiliar crítica para el negocio, como la obtención de datos maestros de la empresa y el cliente, la gestión de la configuración del servicio SUNAT, y las operaciones de E/S de archivos para documentos fiscales.
- **Capa de Acceso a Datos (Inferida):** Se infiere la existencia de una capa de acceso a datos que interactúa con la **Clase Util**. Esta capa sería responsable de la persistencia y recuperación de datos maestros (empresa, clientes) de una base de datos subyacente. Adicionalmente, Util gestiona el acceso a datos relacionados con el almacenamiento de los archivos XML de las facturas generadas y los ZIP de los Comprobantes de Recepción (CDR) de SUNAT en el sistema de archivos.
- **Capa de Integración con Servicios Externos:** Esta capa gestiona la comunicación con entidades externas y servicios especializados.
  - La **Librería Greenter** constituye el componente principal de esta capa, encapsulando la complejidad de la comunicación con los servicios web de la SUNAT (firma digital, protocolos de comunicación).
  - El **Endpoint de SUNAT** (o del intermediario beatose.herokuapp.com) representa el servicio externo final con el que el sistema interactúa para la validación y registro de los comprobantes electrónicos.

## Patrones de Diseño Aplicados

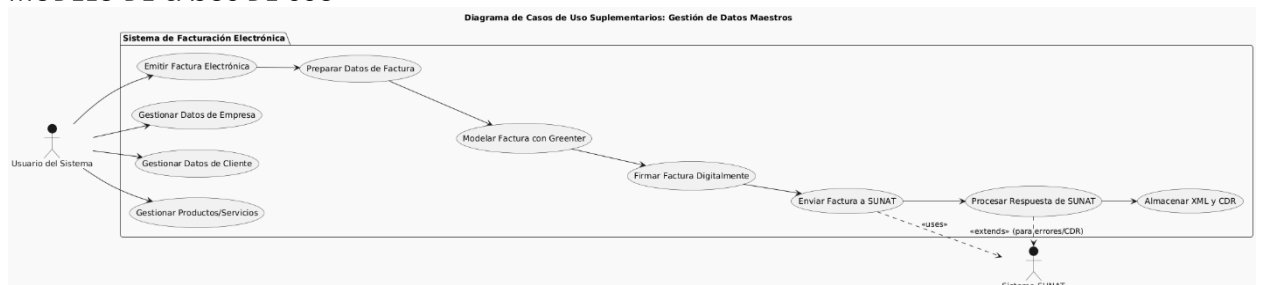
El análisis revela la aplicación de patrones de diseño que contribuyen a la estructura y funcionalidad del sistema:

- **Modelo-Vista-Controlador (MVC) - Aplicación Parcial:** Si bien el sistema no se presenta como un *framework* MVC completo, se observan elementos clave:
  - **Modelos:** Claramente representados por las clases de Greenter\Model (Invoice, SaleDetail, etc.), que encapsulan la información estructurada del documento fiscal.
  - **Controlador:** El script factura-beatose.php asume el rol de un controlador, orquestando el flujo desde la preparación de la factura hasta la gestión de la respuesta del servicio externo. La ausencia de una "Vista" explícita en el fragmento sugiere que la salida (errores, confirmaciones) podría manejarse en la consola o ser integrada en una capa de presentación superior.
- **Singleton:** La clase Util implementa el patrón **Singleton** (Util::getInstance()). Este patrón garantiza que solo exista una única instancia de Util a lo largo de la ejecución del programa. Esta elección de diseño es adecuada para componentes que gestionan recursos compartidos o globales, como configuraciones del sistema, credenciales o conexiones a servicios, asegurando una gestión centralizada y consistente.
- **Fluent Interface (Method Chaining):** El uso extensivo de encadenamiento de métodos (ej., \$invoice->setUblVersion(...)->setFecVencimiento(...)->...) es un patrón que mejora la legibilidad del código al permitir que múltiples operaciones sobre un mismo objeto se realicen de manera concisa y consecutiva.

## Diagrama de arquitectura del sistema.



## 7. MODELO DE CASOS DE USO



## 8. Análisis Estático

Como primer paso realizamos la configuración del entorno en el que se puede desplegar el sistema pa poder utilizarlo en modo local, basándonos en el requerimiento del código alojado en el repositorio de GitHub comprobamos que necesitamos php en sus versiones actualizadas y configuradas con las extensiones soap, y openssl activadas para lo cual se modifican en el archivo php init de nuestro servidor en este caso XAMPP

```

php: Bloc de notes
Archivo Edición Formato Ver Ayuda
;extension=pdo_mysql
;extension=pdo_sqlite
;extension=pgsql
;extension=shmop

; The MIBS data available in the PHP distribution must be installed.
; See https://www.php.net/manual/en/snmp.installation.php
;extension=snmp

extension=soap
extension=sockets
extension=sodium
extension=sqlite3
extension=tidy
extension=xsl
extension=zip

;zend_extension=opcache

; Module Settings ;
;
;
;

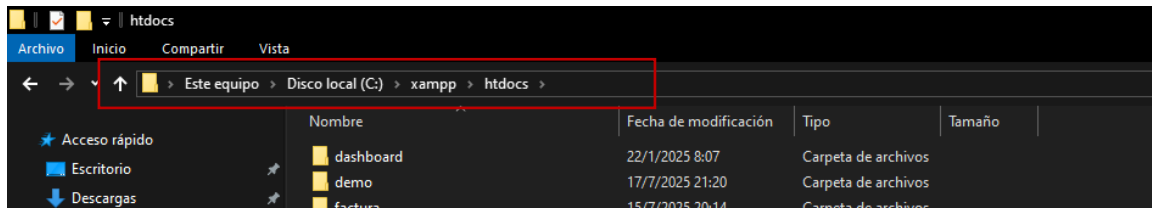
php: Bloc de notes
Archivo Edición Formato Ver Ayuda
extension=openssl
extension=pdo_firebird
extension=pdo_mysql

Buscar
Buscar: extension=openssl
Dirección
Cancelar
Cancelar

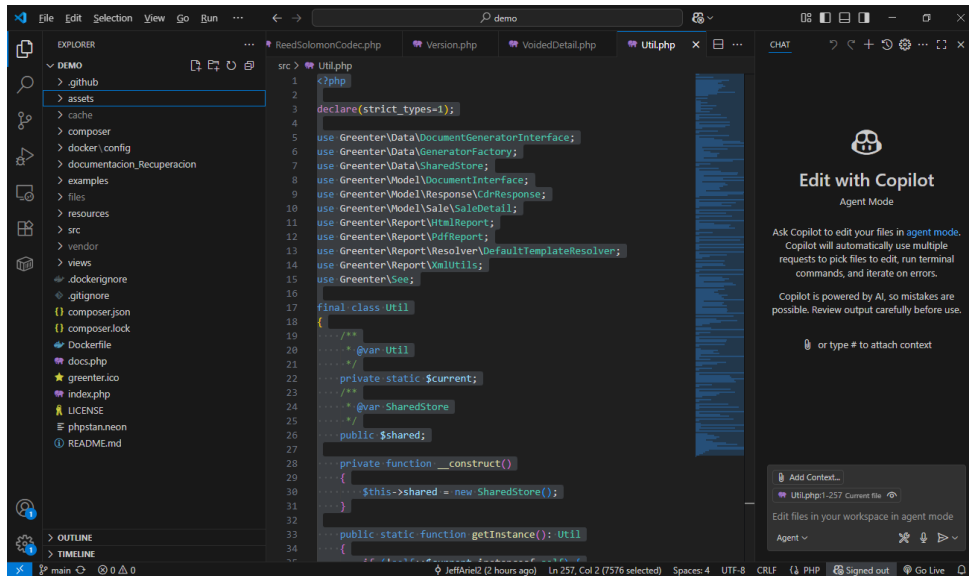
;extension=snmp
extension=soap
extension=sockets
extension=sodium
extension=sqlite3
extension=tidy
extension=xsl
extension=zip

;zend_extension=opcache
  
```

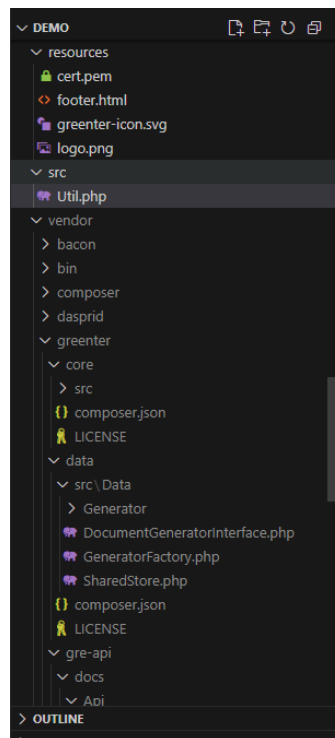
Una vez configurado el servidor de XAMPP en el que queremos correr el aplicativo procedemos a clonar nuestro repositorio en la carpeta del servidor. nuestro directorio clonado se llamará demo



Para el análisis del código fuente y también de hemos elegido el IDE Visual studio Code para lo cual copiamos nuestra carpeta a nuestro IDE para poder analizar de esta manera el sistema en un análisis estático.

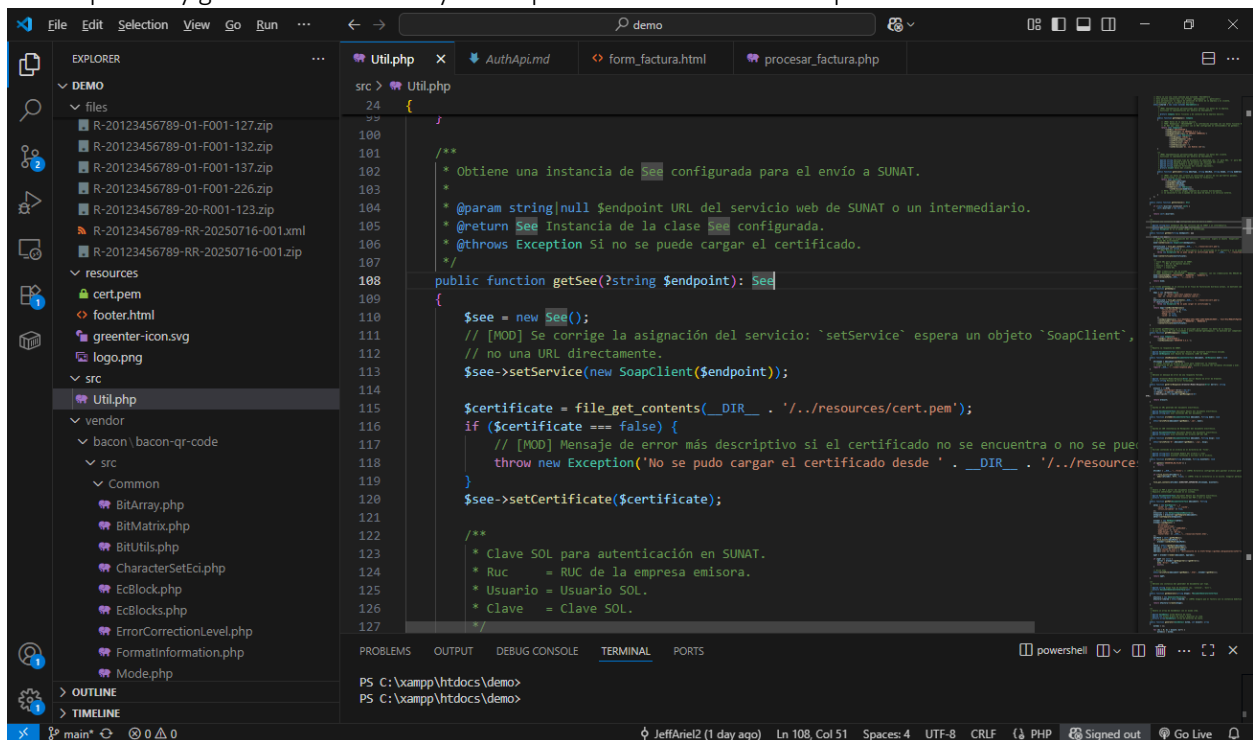


Observando la estructura del directorio nos podemos fijar que tiene muchos documentos de diferente formato lo cual podemos deducir que el sistema tiene lenguajes de JavaScript, PHP, Html,CSS, entre otros,



Una vez integrado el directorio a nuestro ide porcedemos a hacer una análisis breve de los ficheros de los cuales se puede deducir los ficheros mas importantes par asu funcionamiento como son útil.ph Este archivo se encarga de la gestión global y proporciona una única instancia (con el patron sigleton) para acceder a las configuración y servicios de manera consistente en toda la aplicacionademas

centraliza la lógica para obtener y proporcionar los datos de la empresa emisora y los datos dinámico del cliente que proviene de la entrada del usuario, gestiona también al coenxcion con la api de SUNAT y con el certificado digital y las credenciales SOL necesarias para al autenticación, también maneja los archivos crea reportes y gestiona los errores y las respuestas obtenidad de la api de SUNAT



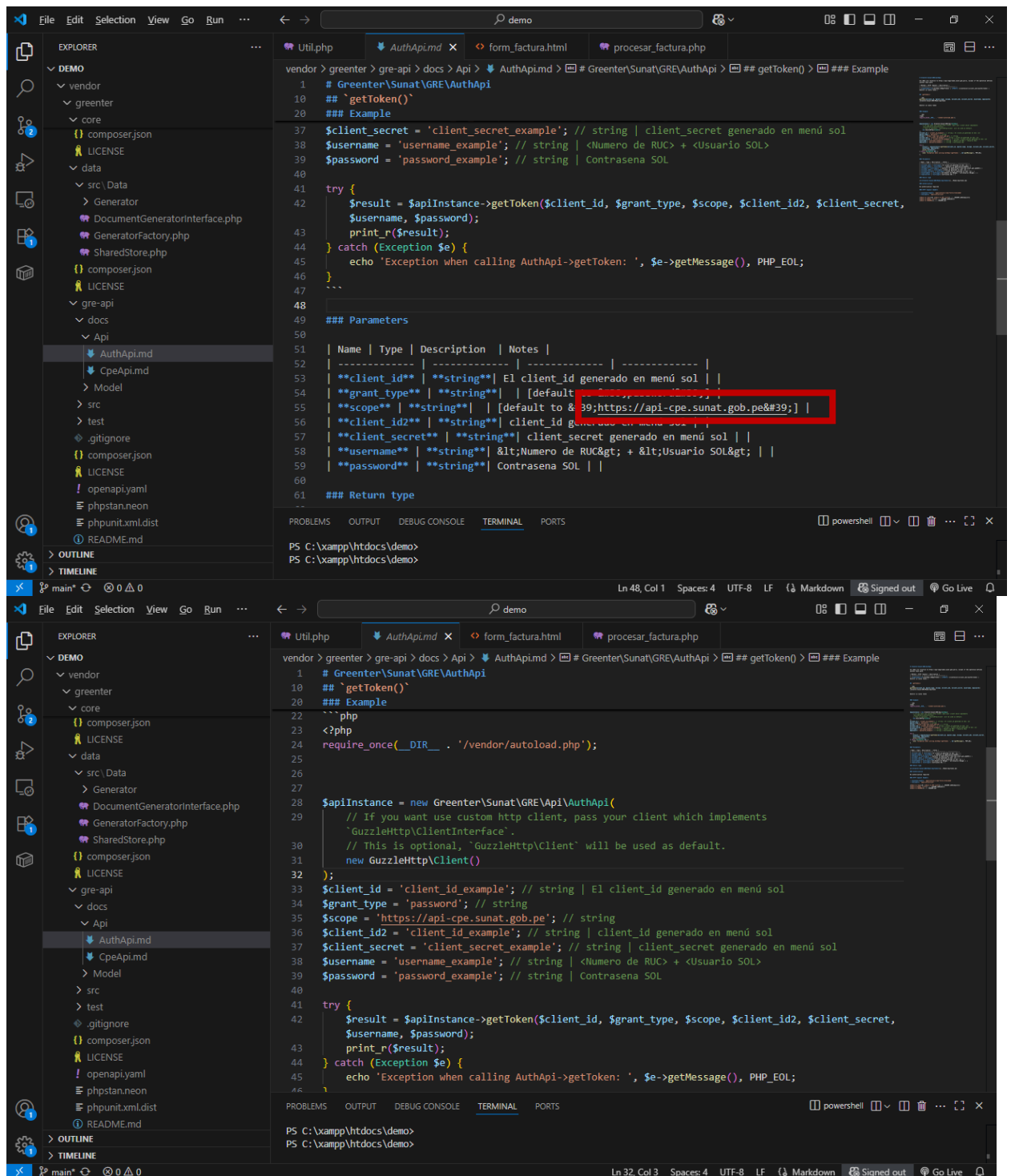
```
src > Util.php
24 {
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127

/**
 * Obtiene una instancia de See configurada para el envío a SUNAT.
 *
 * @param string|null $endpoint URL del servicio web de SUNAT o un intermediario.
 * @return See Instancia de la clase See configurada.
 * @throws Exception Si no se puede cargar el certificado.
 */
public function getSee(?string $endpoint): See
{
    $see = new See();
    // [MOD] Se corrige la asignación del servicio: `setService` espera un objeto `SoapClient`,
    // no una URL directamente.
    $see->setService(new SoapClient($endpoint));

    $certificate = file_get_contents(__DIR__ . '/../resources/cert.pem');
    if ($certificate === false) {
        // [MOD] Mensaje de error más descriptivo si el certificado no se encuentra o no se puede
        // cargar directamente.
        throw new Exception('No se pudo cargar el certificado desde ' . __DIR__ . '/../resources/cert.pem');
    }
    $see->setCertificate($certificate);

    /**
     * Clave SOL para autenticación en SUNAT.
     * Ruc = RUC de la empresa emisora.
     * Usuario = Usuario SOL.
     * Clave = Clave SOL.
     */
}
```

también podemos nombrar como archivo importarte el archivo Authapi.md  
en este archivo se define una el api e cliente que interactúa con el servidor de seguridad de SUNAT, para la generación de e tokens de acceso. Esto esa diseñado para obtener un token de autenticación del servicio de seguridad de SUNAT. también presenta URI relativas al api de seguridad de seguridad de sunat esto especifica el endpoint para la autenticación y a la gestión de tokens.  
AuthApi es la interfaz programática para obtener los tokens de acceso de SUNAT, lo cual es un paso fundamental en el proceso de integración con los nuevos servicios web de facturación electrónica que implementan OAuth2.

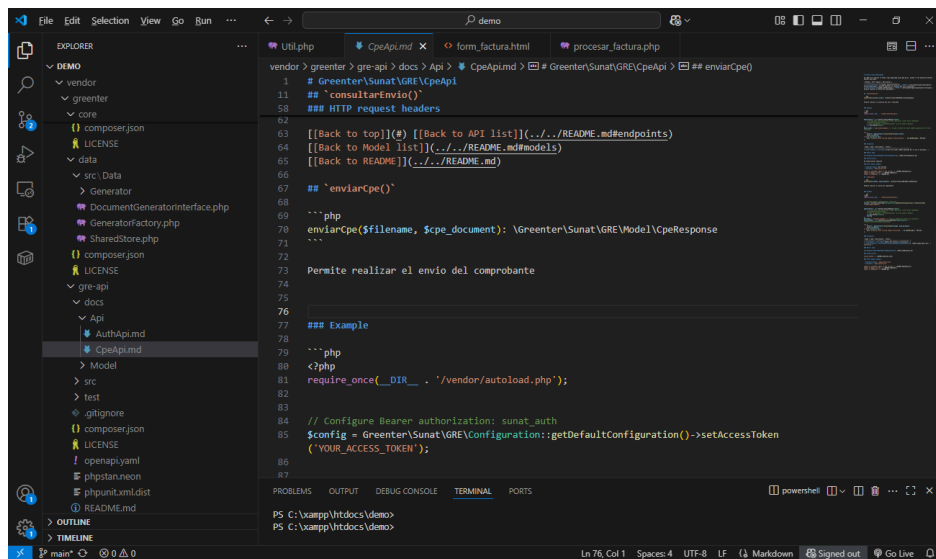


leyendo el código podemos identificar archivos que se están utilizando como referenciados como son `autoload.php`, `ApiToken.md`, `README.md#endpoints`, `README.md#models`, `README.md`. y sirven para facilitar la navegación dentro de la extensa documentación de la API.

La clase `CpeApi.md` actúa como la interfaz principal para interactuar con los servicios web de SUNAT en el contexto de los Comprobantes de Pago Electrónicos (CPE) bajo el nuevo esquema de autenticación OAuth2. Su funcionalidad más importante radica en la capacidad de enviar comprobantes electrónicos a la SUNAT a través del método `enviarCpe()`, el cual espera un objeto `CpeDocument` que encapsula el XML del comprobante, y requiere un token de acceso OAuth2 previamente obtenido para la autenticación. Adicionalmente, esta API proporciona el método `consultarEnvio()`, cuya funcionalidad destacada es permitir verificar el estado de un envío ya realizado a SUNAT utilizando el número de ticket (`numTicket`) recibido en la respuesta del envío inicial. Ambas operaciones utilizan `GuzzleHttp\Client` para las

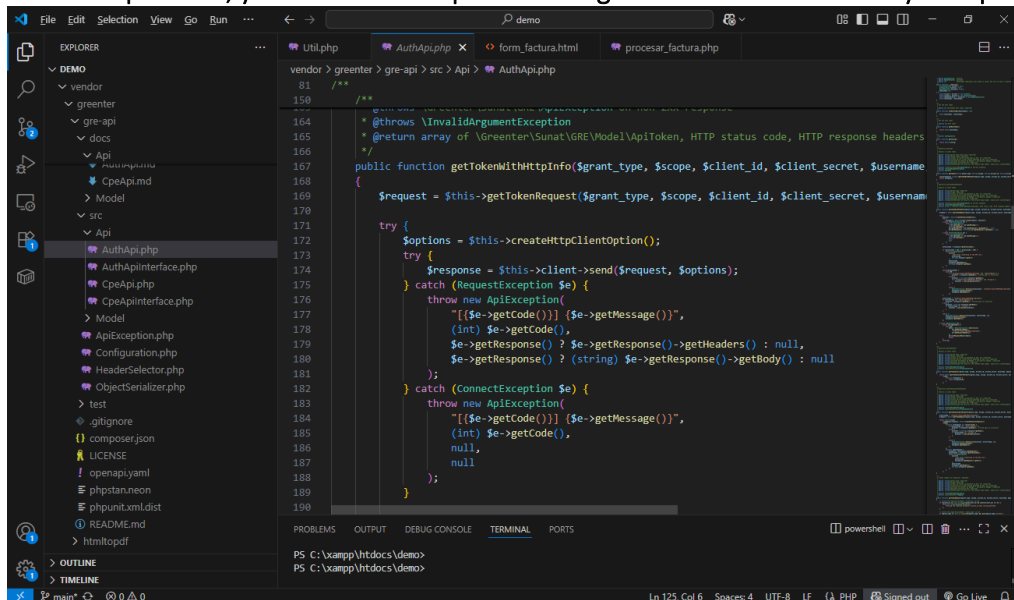


peticiones HTTP y sus URIs base apuntan a los servicios de SUNAT relacionados con la gestión de comprobantes.



```
1  # Greenter\Sunat\GRE\CpeApi
11 ## `consultarEnvio()`
58 ### HTTP request headers
62
63 [[Back to top]](#[[Back to API list]](../README.md#endpoints))
64 [[Back to Model list]](../README.md#models)
65 [[Back to README]](../README.md)
66
67 ## `enviarCpe()`
68
69 ```php
70 enviarCpe($filename, $cpe_document): \Greenter\Sunat\GRE\Model\CpeResponse
71 ...
72
73 Permite realizar el envío del comprobante
74
75
76
77 ### Example
78
79 ```php
80 <?php
81 require_once(__DIR__ . '/vendor/autoload.php');
82
83
84 // Configure bearer authorization: sunat_auth
85 $config = Greenter\Sunat\GRE\Configuration::getDefaultConfiguration()->setAccessToken(
86     'YOUR_ACCESS_TOKEN');
87
```

Este código PHP, generado automáticamente por OpenAPI Generator, define la clase AuthApi que funciona como un cliente de API robusto y estandarizado cuya misión principal es obtener tokens de autenticación OAuth2 del servicio de seguridad de SUNAT. La clase facilita esta tarea a través de su método principal getToken(), el cual se encarga de construir y ejecutar la solicitud HTTP (utilizando GuzzleHttp) con las credenciales de la empresa (client\_id, client\_secret, RUC + Usuario SOL, y Contraseña SOL) y los parámetros de OAuth2 (grant\_type, scope), manejando de forma interna la validación de parámetros, la gestión de errores de conexión y de la API, y la deserialización de la respuesta JSON en un objeto ApiToken para su uso posterior. Además, incluye versiones asíncronas (getTokenAsync()) para operaciones no bloqueantes, y funcionalidades para la configuración del cliente HTTP y la depuración.



```
81 /**
150
164 * @throws \InvalidArgumentException
165 * @return array of \Greenter\Sunat\GRE\Model\ApiToken, HTTP status code, HTTP response headers
166 */
167 public function getTokenWithHttpInfo($grant_type, $scope, $client_id, $client_secret, $username)
168 {
169     $request = $this->getTokenRequest($grant_type, $scope, $client_id, $client_secret, $username);
170
171     try {
172         $options = $this->createHttpClientOption();
173         try {
174             $response = $this->client->send($request, $options);
175         } catch (RequestException $e) {
176             throw new ApiException(
177                 "[{$e->getCode()}] {$e->getMessage()}",
178                 (int) $e->getStatusCode(),
179                 $e->getResponse() ? $e->getResponse()->getHeaders() : null,
180                 $e->getResponse() ? (string) $e->getResponse()->getBody() : null
181             );
182         } catch (ConnectException $e) {
183             throw new ApiException(
184                 "[{$e->getCode()}] {$e->getMessage()}",
185                 (int) $e->getStatusCode(),
186                 null,
187                 null
188             );
189         }
190     }
```

Este código PHP define la interfaz AuthApiInterface, que establece un contrato formal para las clases que interactúan con el servicio de autenticación de SUNAT. Su única función es declarar el método getToken(), especificando que cualquier clase que implemente esta interfaz debe tener un método público con ese nombre, el cual acepta seis parámetros opcionales de tipo cadena (para el tipo de concesión, el alcance, el ID y secreto del cliente, el nombre de usuario y la contraseña) y debe devolver un objeto de tipo \Greenter\Sunat\GRE\Model\ApiToken, asegurando así una estructura consistente para la obtención de tokens de autenticación.

